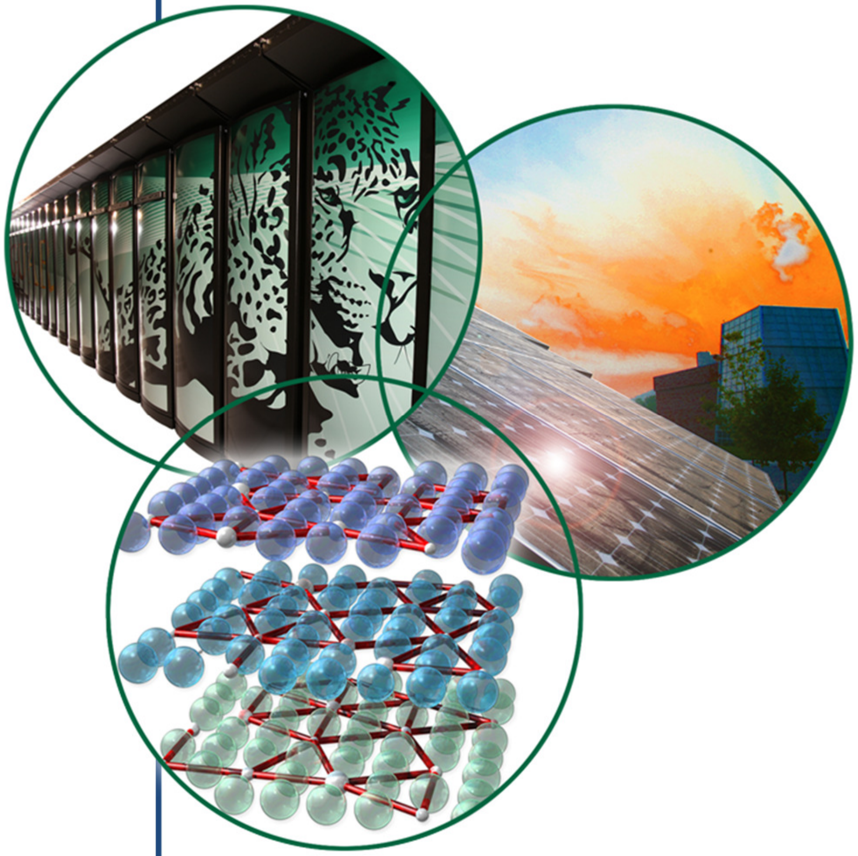
Toward Abstracting the Communication Intent in Applications to Improve Portability and Productivity

Tiffany M. Mintz, Oscar Hernandez, Christos Kartsaklis, David E. Bernholdt, Markus Eisenbach, Swaroop Pophale



U.S. DEPARTMENT OF ENERGY

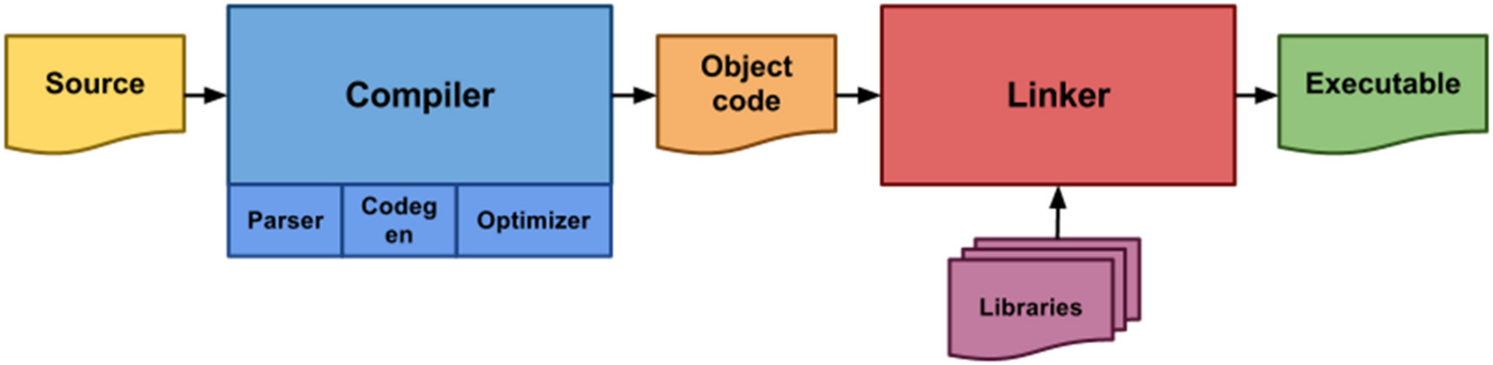
OAK RIDGE NATIONAL LABORATORY
MANAGED BY UT-BATTELLE FOR THE DEPARTMENT OF ENERGY

Content

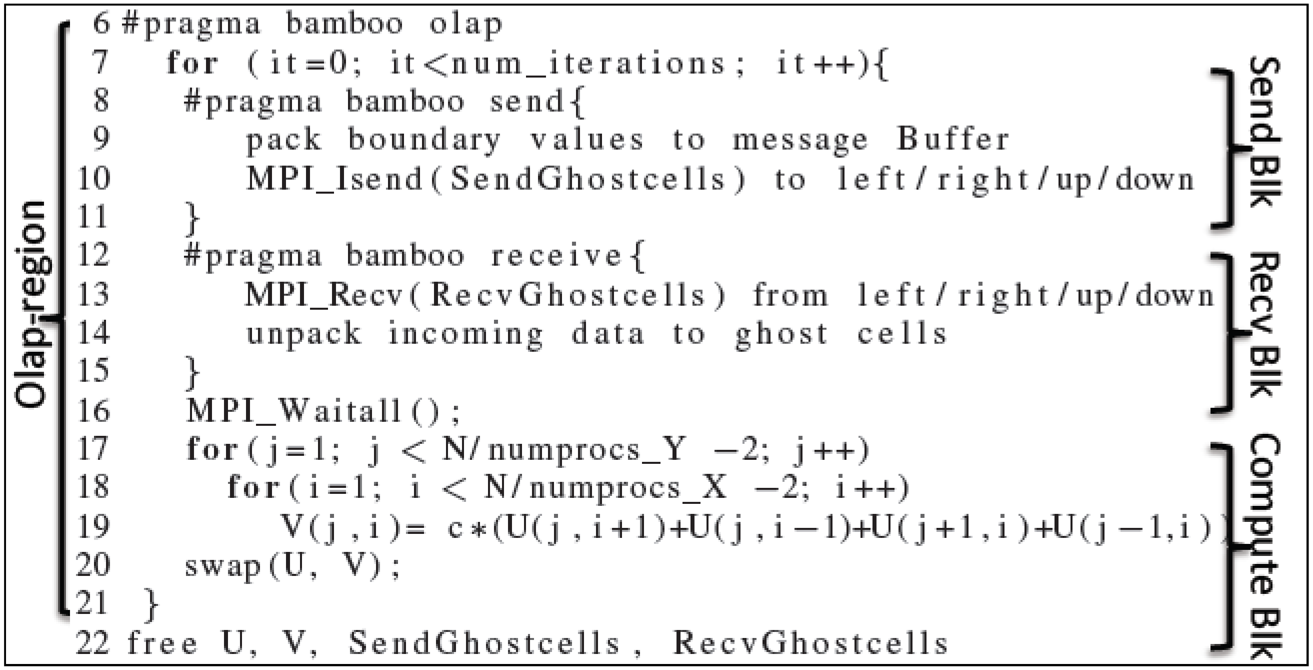
- Motivation
- Related Work
- Approach
- Design & Implementation
- Experimental Results
- Next Steps

Why This Matters

- Communication missing from compiler’s static analysis
  - Opportunities for automatic static optimization lost
- High level abstractions:
  - provide greater portability
  - enhances productivity
  - Easier to maintain



What Others Have Done: Bamboo

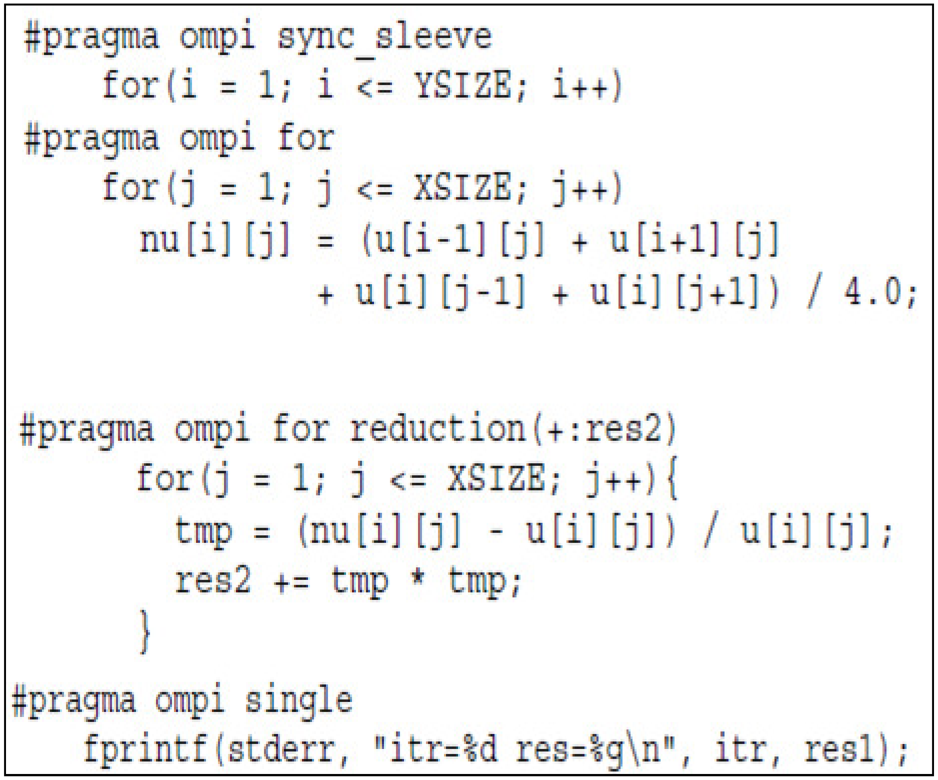


```
#pragma bamboo olap
  for (it=0; it<num_iterations; it++){
   #pragma bamboo send{
      pack boundary values to message Buffer
      MPI_Isend(SendGhostcells) to left/right/up/down
   }
   #pragma bamboo receive{
      MPI_Recv(RecvGhostcells) from left/right/up/down
      unpack incoming data to ghost cells
   }
   MPI_Waitall();
   for(j=1; j < N/numprocs_Y -2; j++)
     for(i=1; i < N/numprocs_X -2; i++)
       V(j,i)= c*(U(j,i+1)+U(j,i-1)+U(j+1,i)+U(j-1,i))
   swap(U, V);
 }
free U, V, SendGhostcells, RecvGhostcells
```

- **Annotations for MPI library calls**
- **Provides mechanism for static analysis and communication/computation overlap**

What Others Have Done: OpenMPI (not the library implementation)

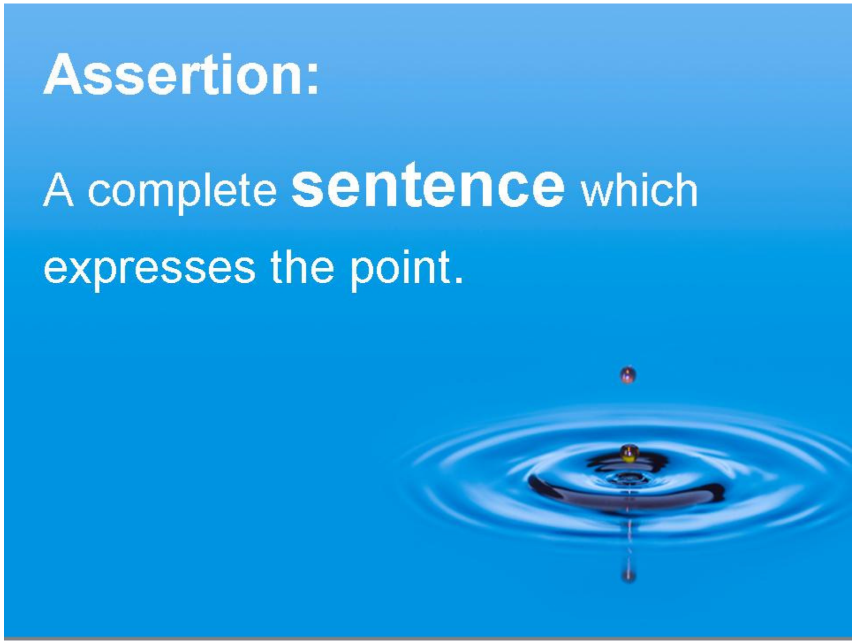
- OpenMP-like directives for incremental parallelization
- Express collective communication



```
#pragma ompi sync_sleeve
    for(i = 1; i <= YSIZE; i++)
#pragma ompi for
    for(j = 1; j <= XSIZE; j++)
      nu[i][j] = (u[i-1][j] + u[i+1][j]
               + u[i][j-1] + u[i][j+1]) / 4.0;


#pragma ompi for reduction(+:res2)
      for(j = 1; j <= XSIZE; j++){
        tmp = (nu[i][j] - u[i][j]) / u[i][j];
        res2 += tmp * tmp;
      }
#pragma ompi single
    fprintf(stderr, "itr=%d res=%g\n", itr, res1);
```

Our Solution

- Use directive language extensions to assert communication
- Translate directives at compile time to communication calls

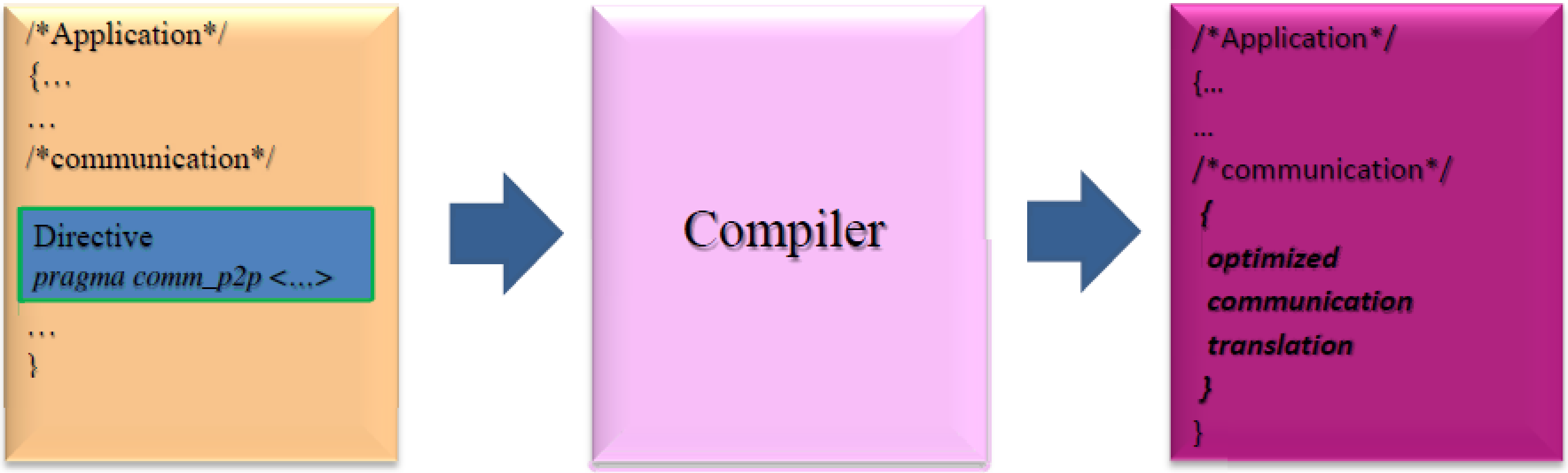


Assertion:

A complete **sentence** which expresses the point.

Asserting Communication

- Directives: comm_p2p, comm_parameters
- Clauses:
  - sender, receiver, sbuf, rbuf
  - sendwhen, receivewhen, count, place_sync, max_comm_iter, target



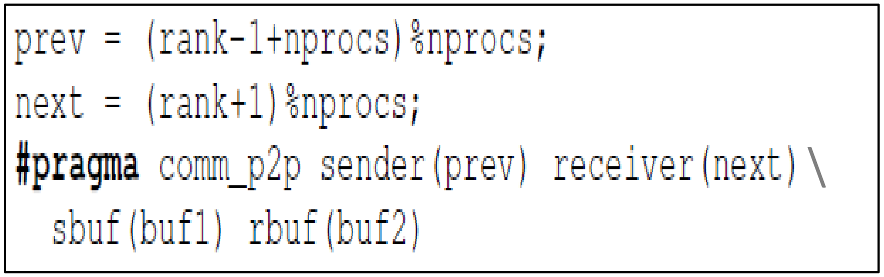
```
/*Application*/
{…
…
/*communication*/
Directive
pragma comm_p2p <…>
…
}
```

Compiler

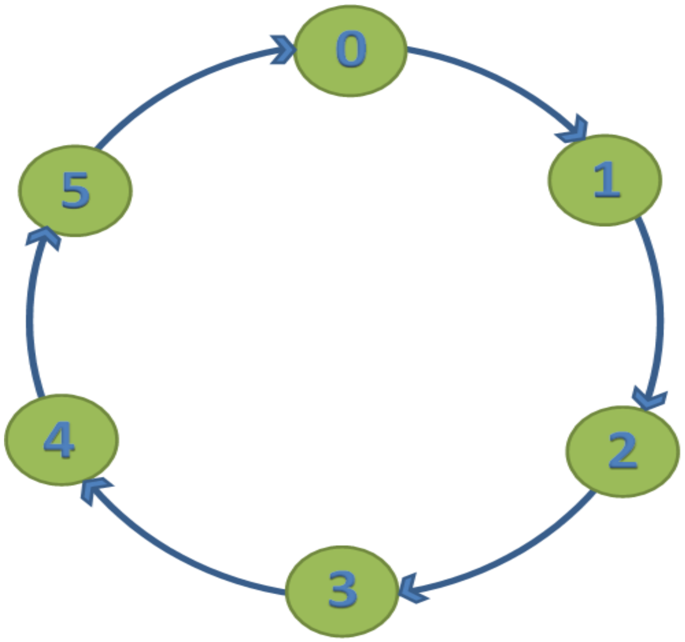
```
/*Application*/
{…
…
/*communication*/
{
 optimized
 communication
 translation
}
}
```

Examples

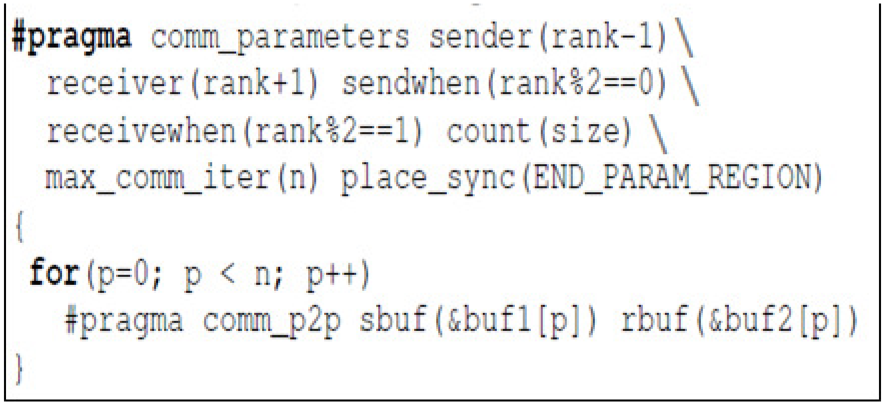
Ring communication pattern



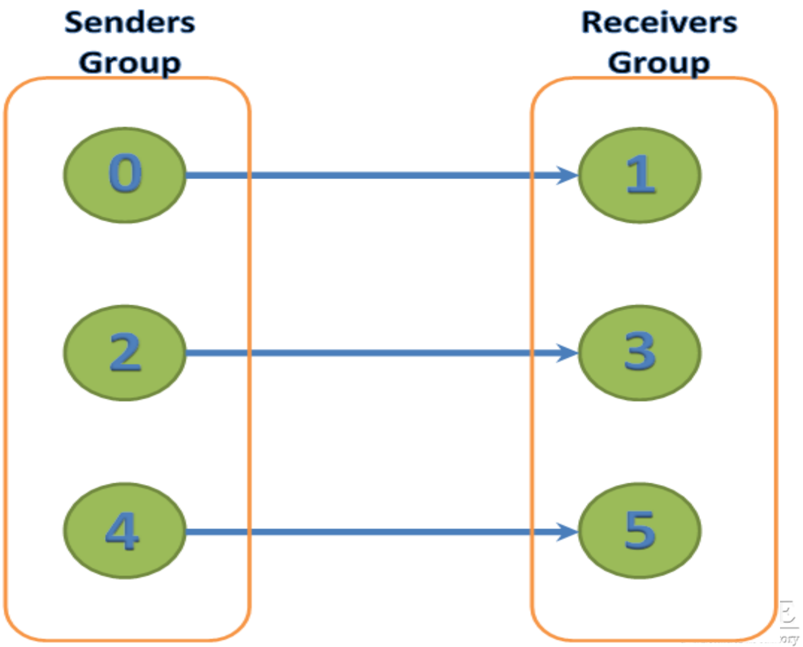
```
prev = (rank-1+nprocs)%nprocs;
next = (rank+1)%nprocs;
#pragma comm_p2p sender(prev) receiver(next)\
  sbuf(buf1) rbuf(buf2)
```



Communication scoping and parameter inheritance

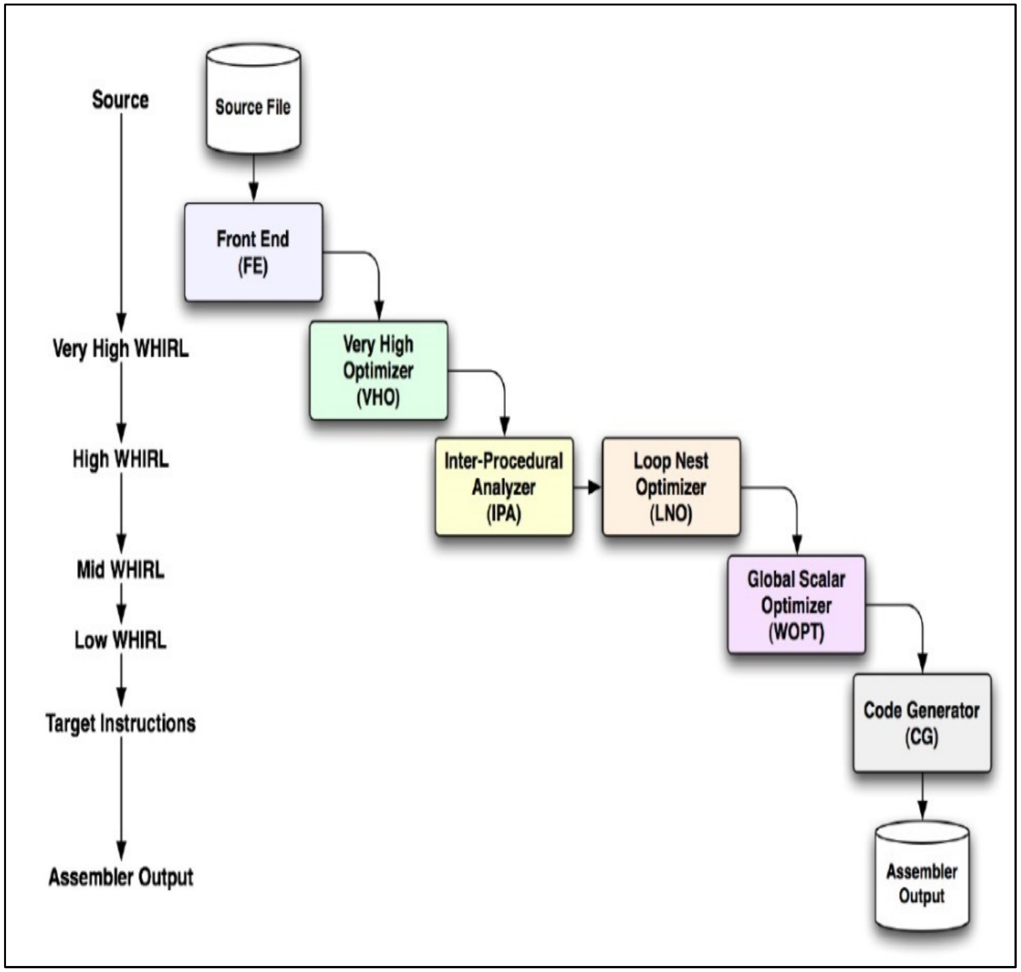


```
#pragma comm_parameters sender(rank-1)\
  receiver(rank+1) sendwhen(rank%2==0) \
  receivewhen(rank%2==1) count(size) \
  max_comm_iter(n) place_sync(END_PARAM_REGION)
{
 for(p=0; p < n; p++)
   #pragma comm_p2p sbuf(&buf1[p]) rbuf(&buf2[p])
}
```



Static Analysis and Optimizations

- Communication scoping
- Communication/computation overlap
- Flexible Implementation
- Data type handling
- Synchronization reduction

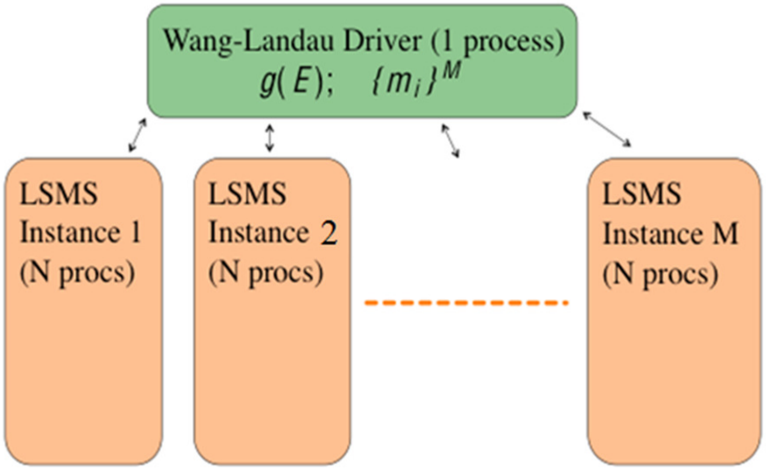


Open64 Compilation Phases

Scientific Application

WL-LSMS

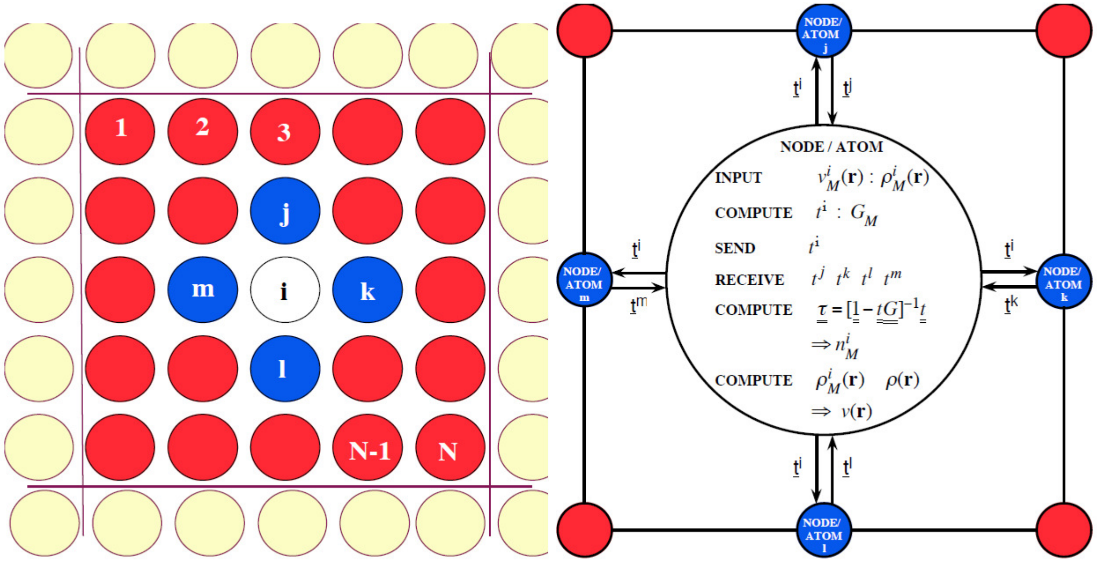
- Wang-Landau (WL)
  - Monte-Carlo calculation
- Locally Self-Consistent Multiple Scattering (LSMS)
  - First principles electronic structure calculation


Wang-Landau Driver (1 process)
$g(E); \quad \{m_i\}^M$
LSMS Instance 1 (N procs)
LSMS Instance 2 (N procs)
LSMS Instance M (N procs)

WL-LSMS Organizational View

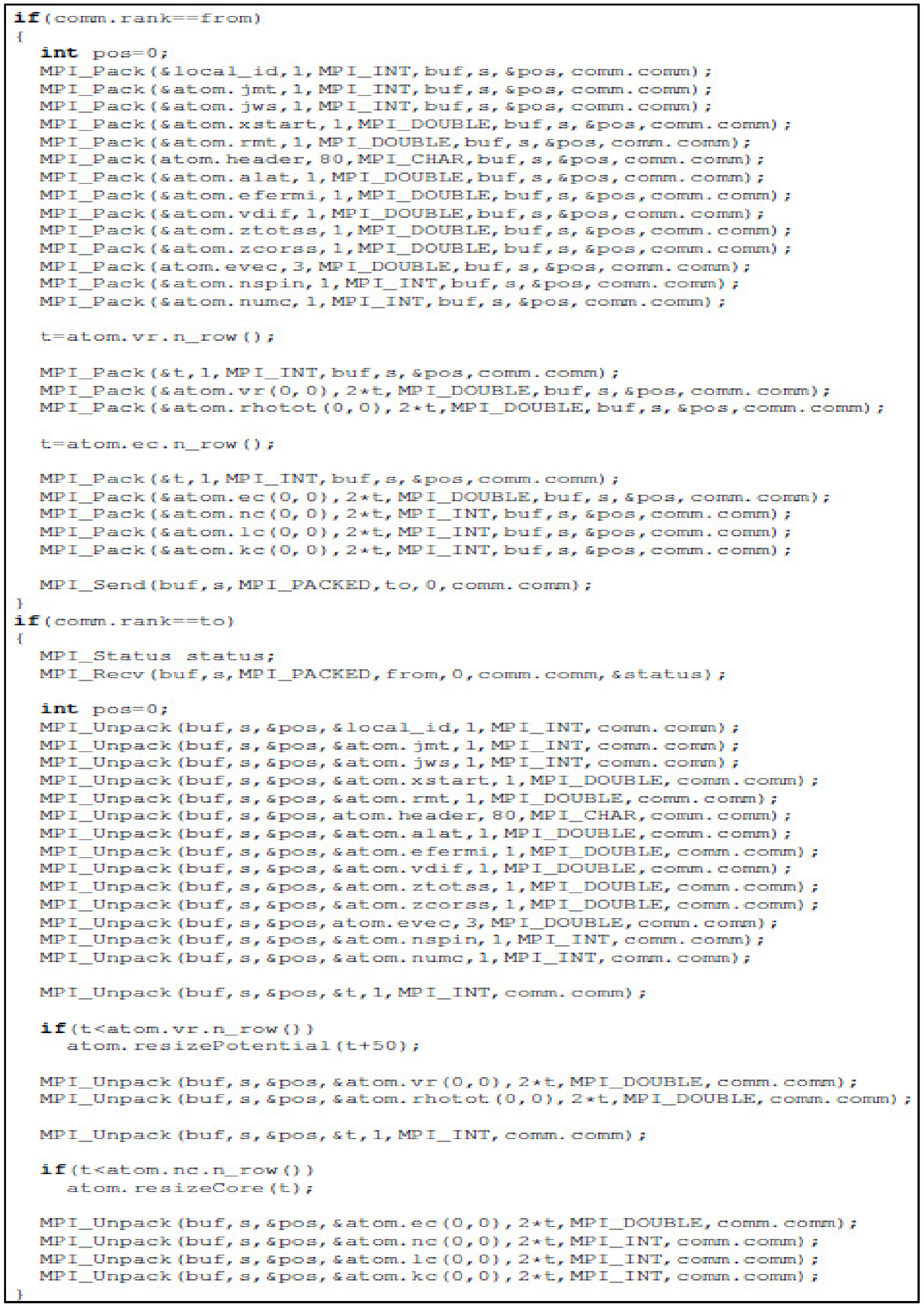
WL-LSMS Communication

- Local Interaction Zone (LIZ)
  - Within each LSMS instance
  - Master-Worker process topology
  - Point-to-point communication


1
2
3
j
m
i
k
l
N-1
N
NODE/ ATOM j
$\underline{t}^i$
$\underline{t}^j$
NODE / ATOM
INPUT $v^i_M(\mathbf{r}) : \rho^i_M(\mathbf{r})$
COMPUTE $t^i : G_M$
SEND t^i
RECEIVE $t^j \; t^k \; t^l \; t^m$
COMPUTE $\underline{\underline{\tau}} = [\underline{\underline{1}} - \underline{\underline{tG}}]^{-1}\underline{\underline{t}}$
$\Rightarrow n^i_M$
COMPUTE $\rho^i_M(\mathbf{r}) \quad \rho(\mathbf{r})$
$\Rightarrow v(\mathbf{r})$
NODE/ ATOM m
$\underline{t}^i$
$\underline{t}^m$
NODE/ ATOM k
$\underline{t}^i$
$\underline{t}^k$
$\underline{t}^i$
$\underline{t}^l$
NODE/ ATOM l

LIZ Communication Pattern

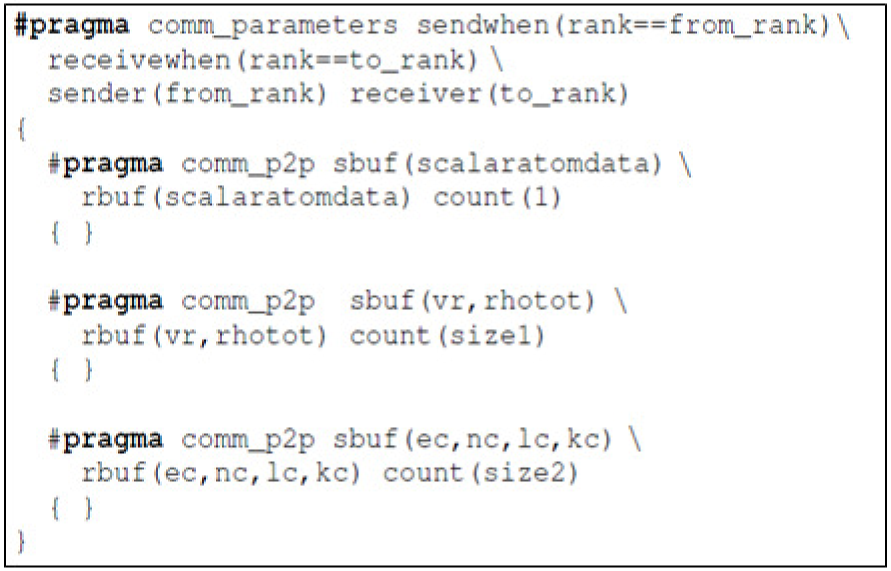
Single Atom Data Communication



```
if(comm.rank==from)
{
  int pos=0;
  MPI_Pack(&local_id,1,MPI_INT,buf,s,&pos,comm.comm);
  MPI_Pack(&atom.jmt,1,MPI_INT,buf,s,&pos,comm.comm);
  MPI_Pack(&atom.jws,1,MPI_INT,buf,s,&pos,comm.comm);
  MPI_Pack(&atom.xstart,1,MPI_DOUBLE,buf,s,&pos,comm.comm);
  MPI_Pack(&atom.rmt,1,MPI_DOUBLE,buf,s,&pos,comm.comm);
  MPI_Pack(atom.header,80,MPI_CHAR,buf,s,&pos,comm.comm);
  MPI_Pack(&atom.alat,1,MPI_DOUBLE,buf,s,&pos,comm.comm);
  MPI_Pack(&atom.efermi,1,MPI_DOUBLE,buf,s,&pos,comm.comm);
  MPI_Pack(&atom.vdif,1,MPI_DOUBLE,buf,s,&pos,comm.comm);
  MPI_Pack(&atom.ztotss,1,MPI_DOUBLE,buf,s,&pos,comm.comm);
  MPI_Pack(&atom.zcorss,1,MPI_DOUBLE,buf,s,&pos,comm.comm);
  MPI_Pack(atom.evec,3,MPI_DOUBLE,buf,s,&pos,comm.comm);
  MPI_Pack(&atom.nspin,1,MPI_INT,buf,s,&pos,comm.comm);
  MPI_Pack(&atom.numc,1,MPI_INT,buf,s,&pos,comm.comm);

  t=atom.vr.n_row();

  MPI_Pack(&t,1,MPI_INT,buf,s,&pos,comm.comm);
  MPI_Pack(&atom.vr(0,0),2*t,MPI_DOUBLE,buf,s,&pos,comm.comm);
  MPI_Pack(&atom.rhotot(0,0),2*t,MPI_DOUBLE,buf,s,&pos,comm.comm);

  t=atom.ec.n_row();

  MPI_Pack(&t,1,MPI_INT,buf,s,&pos,comm.comm);
  MPI_Pack(&atom.ec(0,0),2*t,MPI_DOUBLE,buf,s,&pos,comm.comm);
  MPI_Pack(&atom.nc(0,0),2*t,MPI_INT,buf,s,&pos,comm.comm);
  MPI_Pack(&atom.lc(0,0),2*t,MPI_INT,buf,s,&pos,comm.comm);
  MPI_Pack(&atom.kc(0,0),2*t,MPI_INT,buf,s,&pos,comm.comm);

  MPI_Send(buf,s,MPI_PACKED,to,0,comm.comm);
}
if(comm.rank==to)
{
  MPI_Status status;
  MPI_Recv(buf,s,MPI_PACKED,from,0,comm.comm,&status);

  int pos=0;
  MPI_Unpack(buf,s,&pos,&local_id,1,MPI_INT,comm.comm);
  MPI_Unpack(buf,s,&pos,&atom.jmt,1,MPI_INT,comm.comm);
  MPI_Unpack(buf,s,&pos,&atom.jws,1,MPI_INT,comm.comm);
  MPI_Unpack(buf,s,&pos,&atom.xstart,1,MPI_DOUBLE,comm.comm);
  MPI_Unpack(buf,s,&pos,&atom.rmt,1,MPI_DOUBLE,comm.comm);
  MPI_Unpack(buf,s,&pos,atom.header,80,MPI_CHAR,comm.comm);
  MPI_Unpack(buf,s,&pos,&atom.alat,1,MPI_DOUBLE,comm.comm);
  MPI_Unpack(buf,s,&pos,&atom.efermi,1,MPI_DOUBLE,comm.comm);
  MPI_Unpack(buf,s,&pos,&atom.vdif,1,MPI_DOUBLE,comm.comm);
  MPI_Unpack(buf,s,&pos,&atom.ztotss,1,MPI_DOUBLE,comm.comm);
  MPI_Unpack(buf,s,&pos,&atom.zcorss,1,MPI_DOUBLE,comm.comm);
  MPI_Unpack(buf,s,&pos,atom.evec,3,MPI_DOUBLE,comm.comm);
  MPI_Unpack(buf,s,&pos,&atom.nspin,1,MPI_INT,comm.comm);
  MPI_Unpack(buf,s,&pos,&atom.numc,1,MPI_INT,comm.comm);

  MPI_Unpack(buf,s,&pos,&t,1,MPI_INT,comm.comm);

  if(t<atom.vr.n_row())
    atom.resizePotential(t+50);

  MPI_Unpack(buf,s,&pos,&atom.vr(0,0),2*t,MPI_DOUBLE,comm.comm);
  MPI_Unpack(buf,s,&pos,&atom.rhotot(0,0),2*t,MPI_DOUBLE,comm.comm);

  MPI_Unpack(buf,s,&pos,&t,1,MPI_INT,comm.comm);

  if(t<atom.nc.n_row())
    atom.resizeCore(t);

  MPI_Unpack(buf,s,&pos,&atom.ec(0,0),2*t,MPI_DOUBLE,comm.comm);
  MPI_Unpack(buf,s,&pos,&atom.nc(0,0),2*t,MPI_INT,comm.comm);
  MPI_Unpack(buf,s,&pos,&atom.lc(0,0),2*t,MPI_INT,comm.comm);
  MPI_Unpack(buf,s,&pos,&atom.kc(0,0),2*t,MPI_INT,comm.comm);
}
```

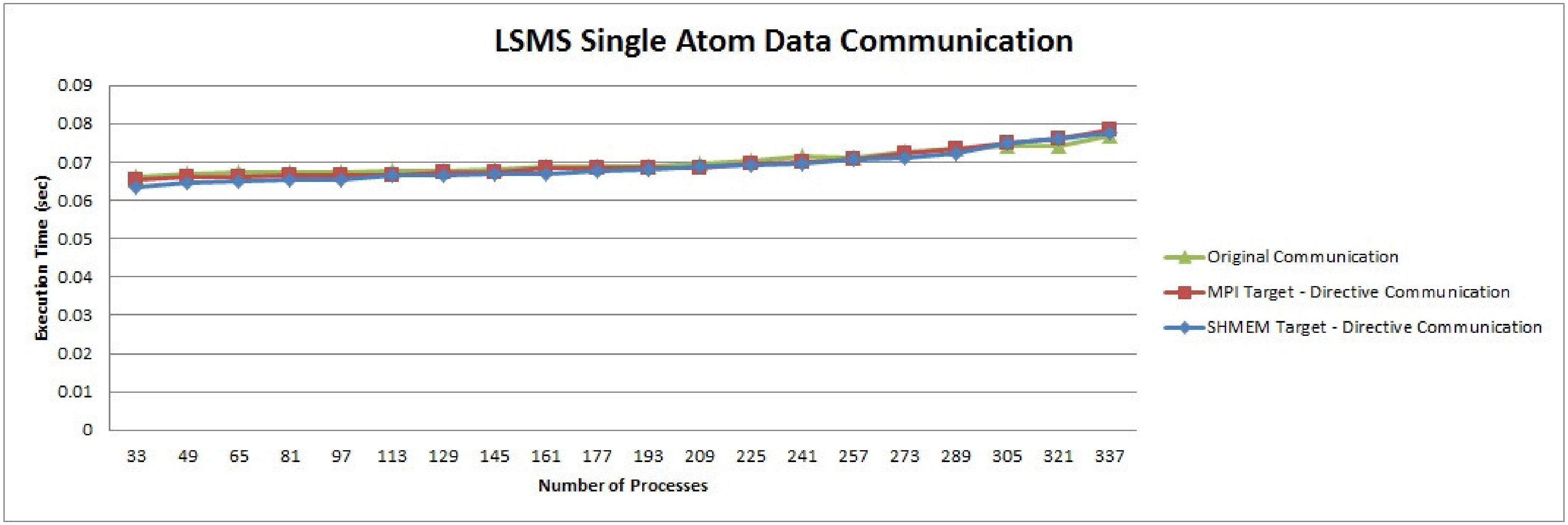
← Original communication source code

Communication using directives ↓



```
#pragma comm_parameters sendwhen(rank==from_rank)\
  receivewhen(rank==to_rank) \
  sender(from_rank) receiver(to_rank)
{
  #pragma comm_p2p sbuf(scalaratomdata) \
    rbuf(scalaratomdata) count(1)
  { }

  #pragma comm_p2p  sbuf(vr,rhotot) \
    rbuf(vr,rhotot) count(size1)
  { }

  #pragma comm_p2p sbuf(ec,nc,lc,kc) \
    rbuf(ec,nc,lc,kc) count(size2)
  { }
}
```

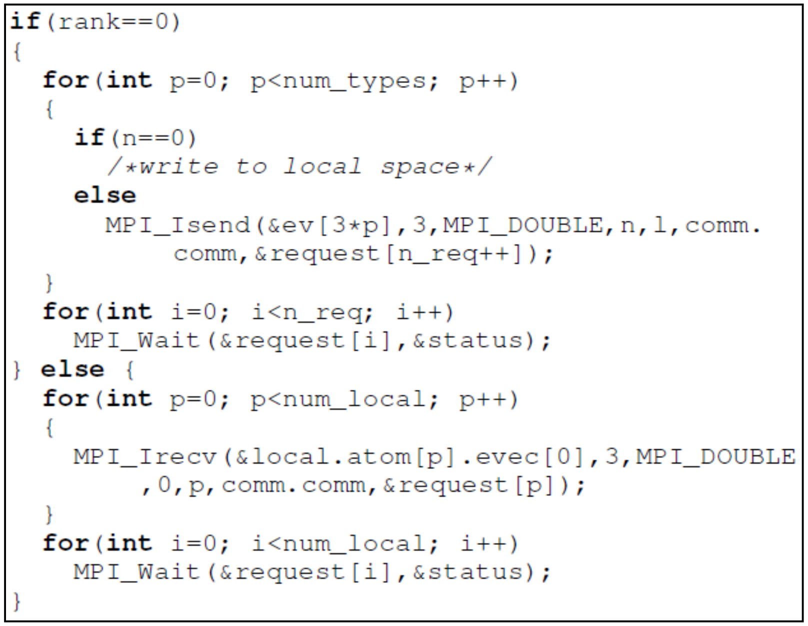
Performance Comparison



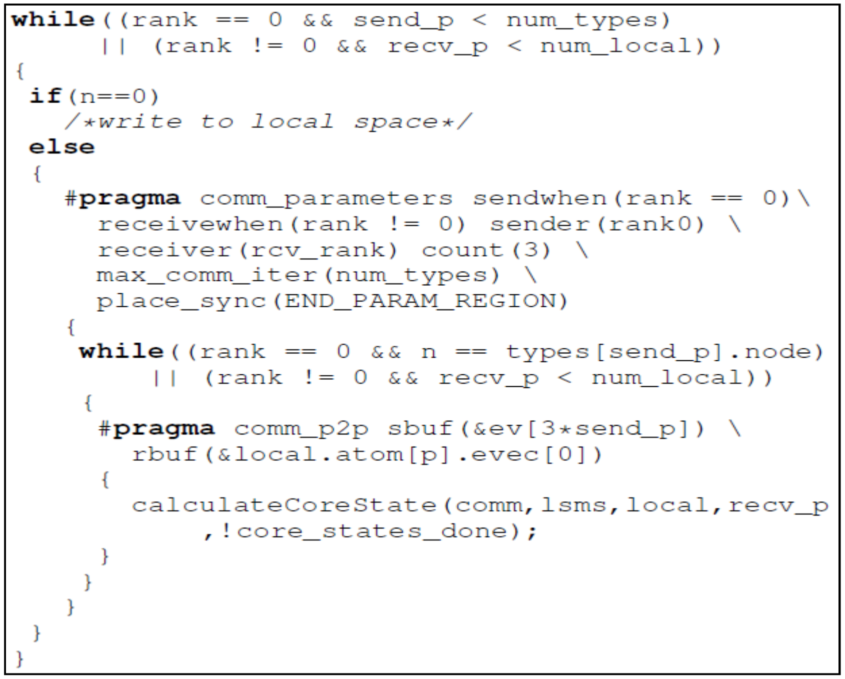
- Experiments using MPI and SHMEM translations
- Performance comparable to original source code

Spin Configurations Communication

Original communication source code



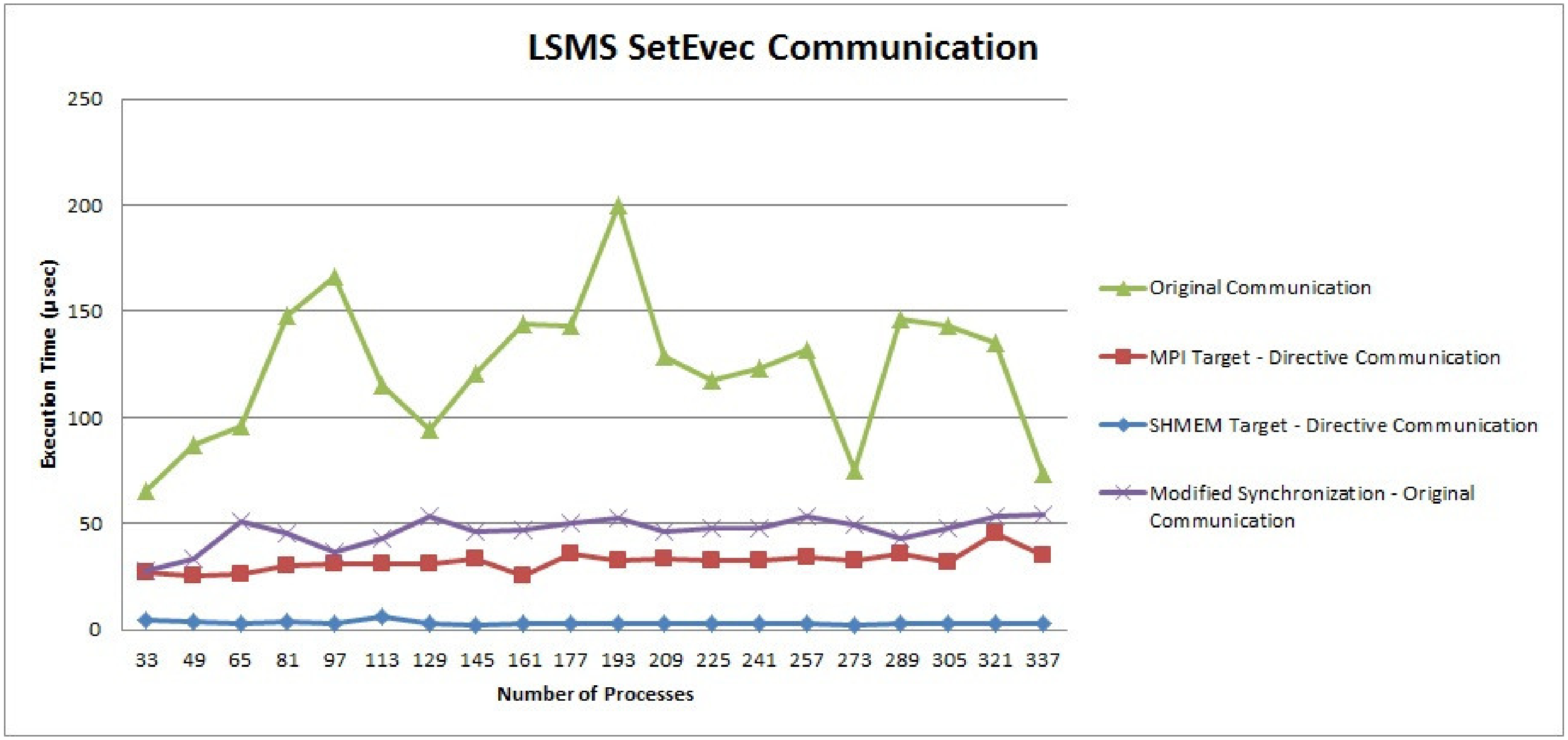
```
if(rank==0)
{
  for(int p=0; p<num_types; p++)
  {
    if(n==0)
      /*write to local space*/
    else
      MPI_Isend(&ev[3*p],3,MPI_DOUBLE,n,1,comm.
          comm,&request[n_req++]);
  }
  for(int i=0; i<n_req; i++)
    MPI_Wait(&request[i],&status);
} else {
  for(int p=0; p<num_local; p++)
  {
    MPI_Irecv(&local.atom[p].evec[0],3,MPI_DOUBLE
        ,0,p,comm.comm,&request[p]);
  }
  for(int i=0; i<num_local; i++)
    MPI_Wait(&request[i],&status);
}
```



```
while((rank == 0 && send_p < num_types)
     || (rank != 0 && recv_p < num_local))
{
 if(n==0)
   /*write to local space*/
 else
 {
   #pragma comm_parameters sendwhen(rank == 0)\
     receivewhen(rank != 0) sender(rank0) \
     receiver(rcv_rank) count(3) \
     max_comm_iter(num_types) \
     place_sync(END_PARAM_REGION)
   {
    while((rank == 0 && n == types[send_p].node)
        || (rank != 0 && recv_p < num_local))
    {
     #pragma comm_p2p sbuf(&ev[3*send_p]) \
       rbuf(&local.atom[p].evec[0])
     {
       calculateCoreState(comm,lsms,local,recv_p
           ,!core_states_done);
     }
    }
   }
 }
}
```

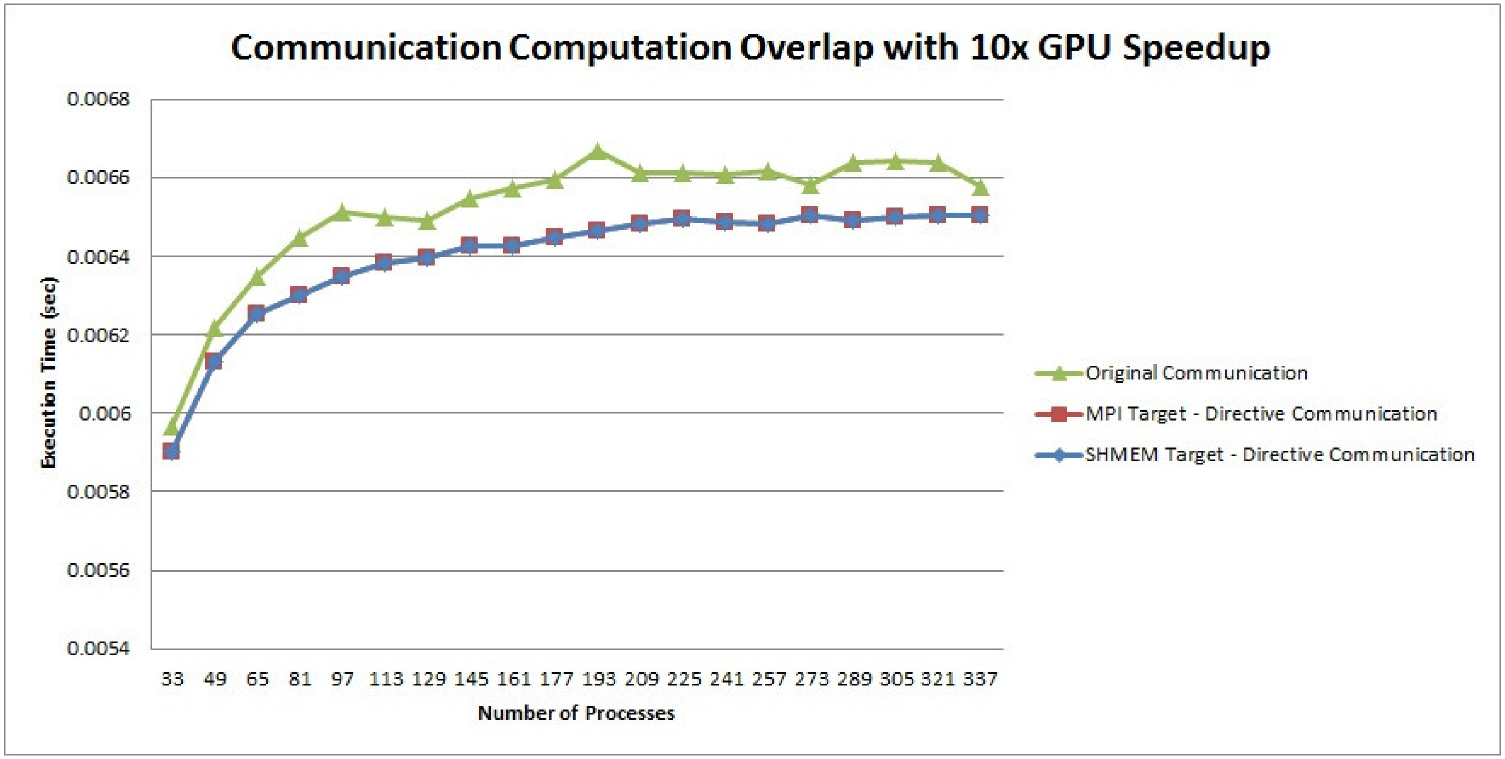
Communication/computation overlap using directives

Communication Comparison



- **Original synchronization caused poor performance**
- **After modifying synchronization:**
  - **MPI translation 1.4x speedup**
  - **SHMEM translation 14.5x speedup**

Communication/Computation Overlap



- **Current Computation/Communication ratio: 19 to 1**
- **Estimate 10x speedup with GPU acceleration**

Looking Forward

- Summary
  - Higher abstraction for message passing communication
  - Communication aware compiler
  - Static analysis and optimization for message passing
- What's Next
  - Develop assertions for many-to-one, one-to-many patterns
  - Extend data flow analysis
  - Implement cost model for automated selection of communication calls

QUESTIONS?